La condición lógica también se puede expresar entre paréntesis:

```
(condicion-logica)? valor-si-verdadero : valor-si-falso;
```

Después de evaluar la condición lógica, se devuelve el valor correspondiente al resultado lógico verdadero o falso. Por ejemplo, dada la edad de una persona, se desea mostrar un mensaje por la consola que indique si es mayor de edad o no.

```
int edad = 16;
String txt;
txt = (edad >= 18) ? "Mayor de edad" : "Menor de edad";
System.out.print(txt);
```

La condición lógica es edad mayor o igual a 18 años. Si es verdadera, el operador devuelve el texto “Mayor de edad”, en caso contrario devuelve “Menor de edad”.

En este ejemplo la variable edad se inicializa a 16, por lo que el mensaje que se muestra por la consola es:

```
Menor de edad
```

Estructuras de repetición

Las estructuras de repetición permiten repetir muchas veces un bloque de sentencias. A estas estructuras también se les conoce como estructuras iterativas o bucles.

Como las estructuras de selección, las estructuras de repetición se pueden combinar y anidar. Es frecuente utilizar una estructura de repetición que contenga un bloque de sentencias que combine otras estructuras de repetición y de selección.

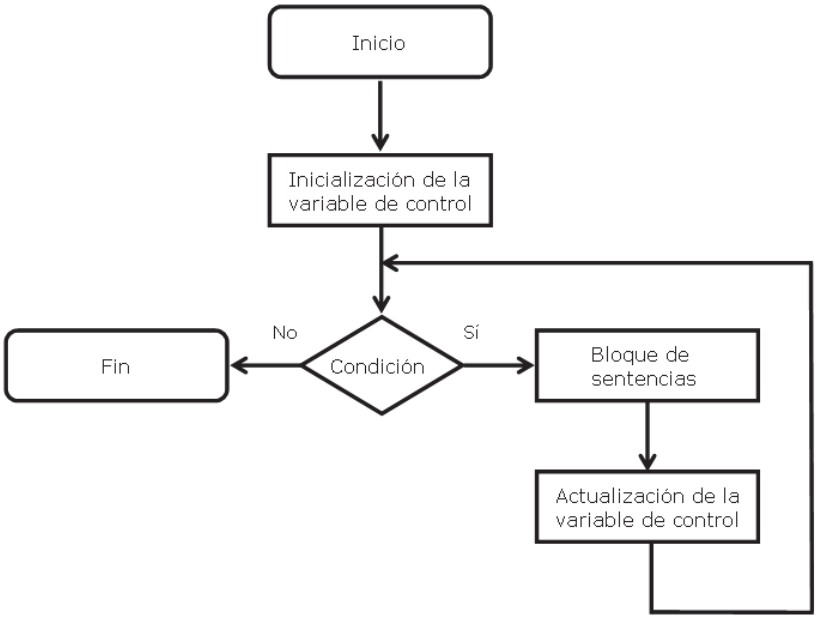
Las estructuras de repetición se componen de cuatro partes: la inicialización, la condición, el bloque de sentencias y la actualización.

- Inicialización. Permite inicializar la estructura iterativa, normalmente consiste en la declaración e inicialización de la variable de control del bucle.
- Condición. Define la condición que se evalúa para ejecutar el bloque de sentencias de la estructura iterativa. Dependiendo del tipo de estructura que se utilice, la condición se comprueba antes o después de realizar cada iteración.
- Bloque de sentencias. Conjunto de sentencias que se ejecutan dentro de la estructura iterativa.
- Actualización. Actualización de la variable de control del bucle. Normalmente se realiza al finalizar la ejecución del bloque de sentencias.

Estructura while

La estructura de repetición `while` repite el bloque de sentencias mientras la condición del `while` es verdadera.

El diagrama de flujo de una estructura `while` muestra que la condición se verifica justo después de inicializar la variable de control. Si el resultado de evaluar la condición por primera es falso, entonces no se ejecuta el bloque de sentencias.



Un `while` tiene la siguiente sintaxis:

```
inicialización;
while (condición) {
   bloque-de-sentencias;
   actualizacion;
}
```

Esta es la sintaxis general. La condición del `while` se escribe obligatoriamente entre paréntesis.

Un `while` no necesariamente requiere inicialización y actualización de una variable de control. En ese caso solo es necesario incluir la condición y el bloque de sentencias:

```
while (condición) {
   bloque-de-sentencias;
}
```

Cuando el programa ejecuta un `while`, lo primero que hace es evaluar la condición. Si es verdadera ejecuta el bloque de sentencias, si es falsa finaliza el `while`.

En cada iteración, cuando finaliza la ejecución del bloque de sentencias se vuelve a evaluar la condición. De nuevo, si es verdadera ejecuta una vez más el bloque de sentencias, si es falsa finaliza el `while`. Cuando esto se produce, el flujo del programa continúa en la sentencia inmediatamente posterior al `while`.

Si la primera vez que se evalúa la condición el resultado es falso, entonces no se ejecuta el bloque de sentencias. Por esta razón, se dice que un `while` se ejecuta cero o más veces. Si la condición siempre es verdadera, entonces el `while` nunca termina y se ejecuta indefinidamente. Esto se conoce como bucle infinito.

El siguiente ejemplo muestra el uso del `while` para calcular la función factorial de un número entero positivo ‘n’.

La función factorial se define:

```
0! = 1
1! = 1
2! = 1 x 2
3! = 1 x 2 x 3
4! = 1 x 2 x 3 x 4
...
n!= 1 x 2 x 3 x 4 x 5 x  ...  x (n-2) x (n-1) x (n)
```

De la definición anterior, se puede calcular el factorial utilizando una estructura repetitiva con una variable de control que empiece en 1 y termine en 'n'. La actualización de la variable de control del `while` suma uno cada vez.

Programa que calcula la función factorial de un número utilizando la estructura `while`.

```
public class FactorialWhile {
   public static void main(String[] args) {
      int n = 5;        // n se inicializa a 5 para calcular 5!
      int factorial = 1 // factorial se inicializa a 1

      int i = 1;        // el valor inicial de i es 1

      while (i <= n) {
         factorial = factorial * i;
         i++;
      }

      System.out.println("El factorial de " + n + " es " +
                         factorial);

   }
}
```

En la expresión `factorial = factorial * i` la variable `factorial` aparece dos veces. Primero se calcula el producto `factorial * i` y después se asigna este resultado a la variable `factorial`. Es por esto que la tabla muestra una columna con los valores de los operandos del producto y otra con el valor final de la variable factorial. La siguiente tabla muestra el proceso de cálculo que se realiza en el `while`.

i	n	factorial * i	factorial
1	5	1 * 1	1
2	5	1 * 2	2
3	5	2 * 3	6
4	5	6 * 4	24
5	5	24 * 5	120

La variable `factorial` se inicializa a 1. En la primera iteración `i` vale 1, se calcula el producto `factorial * i` con los valores 1 * 1 y se asigna 1 a la variable `factorial`. En la segunda iteración `i` vale 2, se calcula `factorial * i` con los valores 1 * 2 y se asigna 2 a la variable `factorial`. En la tercera iteración `i` vale 3, se calcula `factorial * i` con los valores 2 * 3 y se asigna 6 a la variable `factorial`. En la cuarta iteración `i` vale 4, se calcula `factorial * i` con los valores 6 * 4 y se asigna 24 a la variable `factorial`. En la última iteración `i` vale 5, se calcula el producto `factorial * i` con los valores 24 * 5 y se asigna 120 a la variable `factorial`.

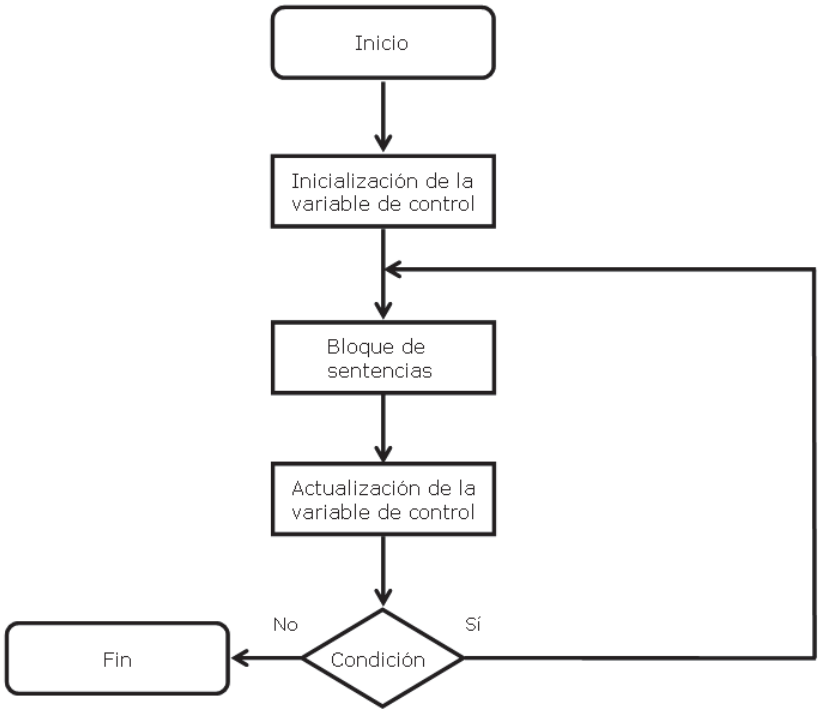
De los resultados de la tabla anterior, se puede observar que no es necesario calcular el producto `factorial * i` en la primera iteración cuando `i` vale 1. Este producto siempre va a dar como resultado 1.

El algoritmo es más eficiente si se elimina la primera iteración, basta con inicializar la variable de control `i` a 2. En este ejemplo, el valor inicial de `i` es 1 para que el algoritmo sea más claro.

Estructura do-while

La estructura de repetición `do-while`e ejecuta el bloque de sentencias al menos una vez. Después comprueba la condición y repite el bloque de sentencias mientras la condición es verdadera.

El diagrama de flujo de una estructura `do-while` muestra que la condición se verifica al final, después de ejecutar el bloque de sentencias la primera vez.



Un `do-while` tiene la siguiente sintaxis:

```
inicialización;
do {
   bloque-de-sentencias;
   actualizacion;
} while (condición);
```

Esta es la sintaxis general. La condición del `do-while` se escribe obligatoriamente entre paréntesis.

Un `do-while` no necesariamente utiliza una variable de control. En ese caso solo es necesario incluir la condición y el bloque de sentencias:

```
do {
   bloque-de-sentencias;
} while (condición);
```

Cuando el programa ejecuta un `do-while`, lo primero que hace es ejecutar el bloque de sentencias y luego evalúa la condición. Si es verdadera, ejecuta de nuevo el bloque de sentencias, si es falsa finaliza el `do-while`.

En cada iteración, cuando finaliza la ejecución del bloque de sentencias se vuelve a evaluar la condición. De nuevo, si es verdadera ejecuta una vez más el bloque de sentencias, si es falsa finaliza el `do-while`. Cuando esto se produce, el flujo del programa continúa en la sentencia inmediatamente posterior al `do-while`.

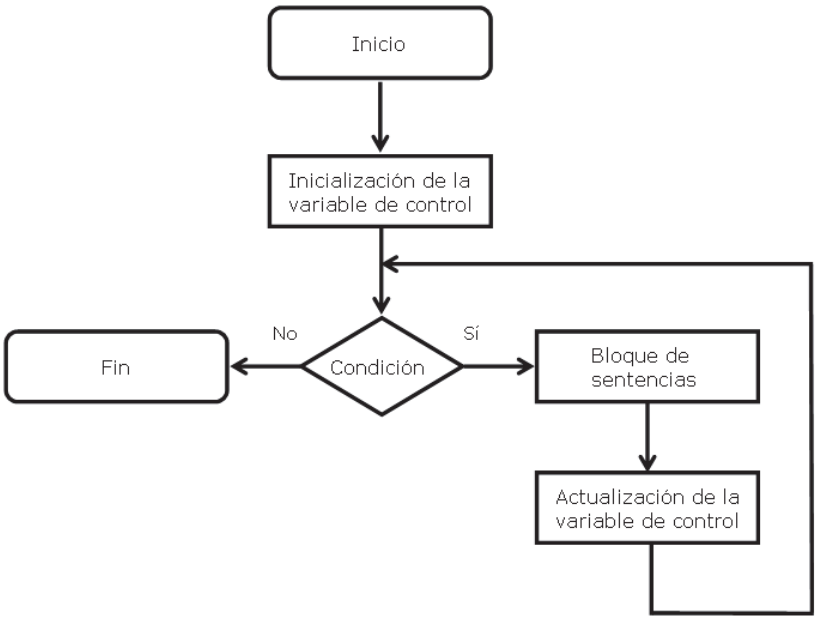
Programa que calcula la función factorial de un número utilizando la estructura `do-while`.

```
public class FactorialDoWhile {
   public static void main(String[] args) {
      int n = 5;         // n se inicializa a 5 para calcular 5!
      int factorial = 1 // factorial se inicializa a 1

      int i = 1;         // el valor inicial de i es 1

      do {
         factorial = factorial * i;
         i++;
      } while (i <= n);

      System.out.println("El factorial de " + n + " es " +
                         factorial);

   }
}
```

Estructura for

La estructura de repetición `for` repite el bloque de sentencias mientras la condición del `for` es verdadera. Un `for` es un caso particular de la estructura `while`. Solo se debe utilizar cuando se sabe el número de veces que se debe repetir el bloque de sentencias.

El diagrama de flujo de una estructura `for` es igual que el de un `while`. Un `for` verifica la condición justo después de inicializar la variable de control. Si el resultado de evaluar la condición por primera es falso, entonces no se ejecuta el bloque de sentencias.



Un `for` tiene la siguiente sintaxis:

```
for (inicialización; condición; actualización) {
   bloque-de-sentencias;
}
```

Cuando el programa ejecuta un `for`, lo primero que hace es evaluar la condición. Si es verdadera ejecuta el bloque de sentencias, si es falsa finaliza el `for`.

En cada iteración, cuando finaliza la ejecución del bloque de sentencias se vuelve a evaluar la condición. De nuevo, si es verdadera ejecuta una vez más el bloque de sentencias, si es falsa finaliza el `for`. Cuando esto se produce, el flujo del programa continúa en la sentencia inmediatamente posterior al `for`.

Programa que calcula la función factorial de un número utilizando la estructura `for`.

```
public class FactorialFor {
   public static void main(String[] args) {
      int n = 5;        // n se inicializa a 5 para calcular 5!
      int factorial = 1 // factorial se inicializa a 1

      for (int i=1; i <= n; i++) {
         factorial = factorial * i;
      }

      System.out.println("El factorial de " + n + " es " +
                         factorial);

   }
}
```

Normalmente la variable de control se declara y se inicializa en la sección de inicialización de la variable. En este ejemplo se hace `int i= 1`, es decir se declara una variable `i` de tipo `int` y se inicializa a 1. La condición del `for` es `i <= n`, la misma que se ha utilizado en el `while` y el `do-while`. Por último, la variable `i` se incrementa en 1 en cada iteración. En el `for`, el `while` y el `do-while` el incremento de `i` se realiza con el operador ++.

Es posible combinar estructuras de selección y estructuras de iteración. Si se define una estructura de repetición dentro de otra, entonces se tiene una estructura de repetición anidada.

El siguiente ejemplo utiliza tres for anidados. ¿Cuántas veces se muestra por la consola el mensaje "Hola Mundo"?

```
public class ForAnidado {
   public static void main(String[] args) {

      for (int i=1; i <= 5; i++)
         for (int j=2; j <= 4; j++)
            for (int k=3; k <= 6; k++)
               System.out.println("Hola Mundo");
   }
}
```

Para saber cuántas veces se imprime el mensaje es necesario saber cuántas veces se repite cada `for`. El for de `i` se repite 5 veces, el `for` de `j` se repite 3 veces y el `for` de `k` se repite 4 veces. Como el `for` de `k` está dentro del `for` de `j` y éste dentro del `for` de `i`, el mensaje se imprime 5x3x4 veces, un total de 60 veces.

Uso de las estructuras de repetición

Es importante utilizar la estructura de repetición más apropiada para cada caso. En general, se recomienda seguir los siguientes criterios:

- El `while` se debe utilizar cuando no se sabe el número de veces que se va a repetir el bloque de sentencias.
- El `do-while` se debe utilizar cuando el bloque de sentencias se debe ejecutar al menos una vez.
- El `for` se debe utilizar cuando se sabe el número de veces que se va a repetir el bloque de sentencias. Un `for` es útil cuando se conoce el valor inicial para la variable de control del bucle y además es necesario utilizar una expresión aritmética para actualizar esta variable.

Ejemplo de uso de while. Utilice una estructura `while` para determinar mediante restas sucesivas si un número entero positivo es par.

Para saber si un número entero es par es necesario restar 2 sucesivamente mientras el número sea mayor o igual a 2. Si después de realizar las restas el número es cero, el número es par, si no, es impar.

```
public class NumeroParImpar {
   public static void main(String[] args) {

      // este programa verifica si un número positivo
      // es para o impar

      int numero = 12;    // el valor inicial del número

      while (numero >= 2) {
         numero = numero - 2;
      }

      if (numero == 0)
         System.out.println("El número es par");
      else
         System.out.println("El número es impar");

   }
}
```

Ejemplo de uso de do-while. Utilice una estructura `do-while` que muestre por la consola números enteros aleatorios entre 0 y 100 hasta que salga el número 50.

Para calcular un número aleatorio se utiliza el método `random()` de la clase `Math`. Este método devuelve un valor de tipo `double` entre 0 y 1. Este resultado se multiplica por 100 para que el valor esté en el rango entre 0 y 100. Antes de asignar el resultado a la variable número se convierte a un valor entero utilizando `(int)`.

El `do-while` se ejecuta al menos una vez y muestra los números aleatorios calculados mientras el número sea diferente de 50.

```
public class NumerosAleatorios {
   public static void main(String[] args) {

      // este programa muestra números enteros aleatorios
      // entre 0 y 100 hasta que sale el 50

      do {
         numero = (int) (100 * Math.random());
         System.out.println("Número aleatorio: " + numero);
      } while (numero != 50);

   }
}
```

Ejemplo de uso de for. Utilice una estructura `for` para calcular la función potencia de un número entero positivo utilizando productos. La potencia se calcula como el producto de la base repetido tantas veces como el valor del exponente.

```
potencia = base x base x base x base x base x ... x base
```

Inicialmente, el valor de la variable potencia es 1 porque cualquier número elevado a la potencia cero es 1.

```
public class PotenciaFor {
   public static void main(String[] args) {

      // este programa calcula 2^10

      int base = 2;        // base
      int exponente = 10; // exponente al que se eleva la base
      int potencia = 1;    // potencia se inicializa a 1
                           // porque x^0 = 1

      for (int i=1; i <= exponente; i++) {
         potencia = potencia * base;
      }

      System.out.println("La potencia es " + potencia);

   }
}
```